

A Network GPIB Controller

Project Number eZ2963

Introduction

Test equipment such as oscilloscopes, power supplies, signal generators, etc. are often used in manufacturing, compliance, and other applications where their behavior needs to be controlled or automated to perform tasks. Back in the 1960's, Hewlett-Packard recognized this need and invented a generic bus protocol that allows such equipment to be easily connected together and controlled. This protocol is commonly known today as the General Purpose Interface Bus (GPIB), although it also known by its standardized version, IEEE 488.1.

GPIB is a multipoint, 8-bit parallel bus that uses a three-wire handshake to acknowledge each data byte. The bus is organized in a master-slave arrangement where there is at least one controller that is responsible for directing bus communication and a number of talkers that send data and listeners that receive data. Every device on the bus can serve any combination of these three roles. The bus allows a total of 15 devices with up to 2m of separation between each device and a maximum bus length of 20m. The maximum nominal transfer rate of the bus is 1 MB/sec though there are non-standard enhancements that can allow it to operate faster. GPIB has proven to be a very popular protocol and it is supported on a large percentage of test equipment.

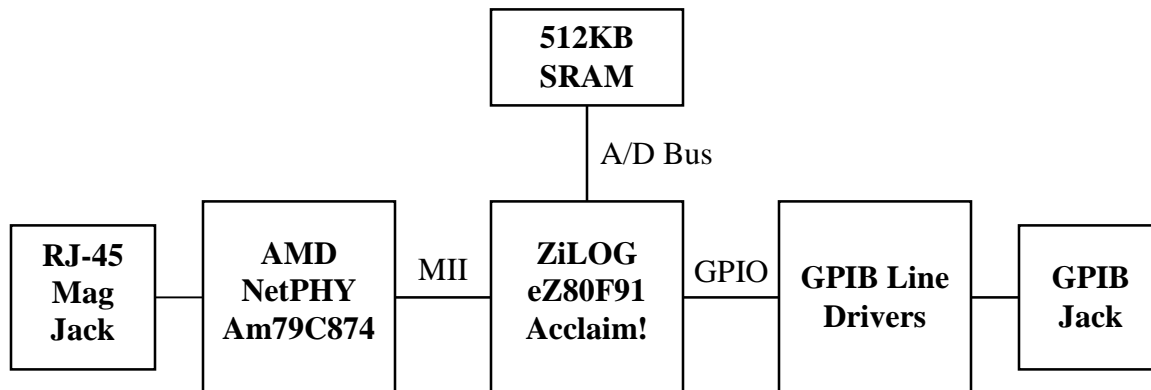
Although GPIB has proven to be very popular, the rise of the Internet age has made network-enabled electronics all the rage and this influence has also affected the test equipment industry. More and more test equipment is including built-in Ethernet interfaces with embedded web servers or other network control interfaces in addition to or completely replacing the GPIB interface. The main advantages of network-enabled control are that the cabling costs are much lower and that there are no inherent distance limitations. In addition, embedding a web server or other client application allows the equipment to be controlled by any available PC with a web browser without having to install the client software on the PC first.

There exist a few devices that will allow for control of equipment with only a GPIB interface over an Ethernet connection. These devices suffer from two major drawbacks. First, they are unnecessarily expensive which can put them out of reach for individuals and small companies. Second, they do not embed the client application with which to control the GPIB equipment so the user must still install software on each PC. Our network GPIB controller solves both of these drawbacks by providing a low-cost solution with the ability to easily embed client applications.

The network GPIB controller uses the eZ80 processor to integrate a number of critical functions into a single device to produce a low-cost yet feature-rich solution. The heart of the controller is the GPIB driver which implements a fully compliant GPIB bus controller in software by using the eZ80's external GPIO lines. The eZ80 processor chosen also incorporates an embedded Ethernet MAC which helped reduce both the cost and complexity.

The server software in the controller is built on top of the freely available ZTP TCP/IP protocol stack. We selected the XML-RPC remote procedure call protocol to be the main means of control between the remote client applications and the controller. XML-RPC is a simple protocol based on XML and is transported via HTTP. Its main advantages are that it is lightweight and well-supported under a number of programming environments.

In order to demonstrate the ability to embed a client application in the controller and use it to control GPIB equipment, we also implemented a GUI application in Java for the Tektronix TDS-210 oscilloscope. This application has the ability to graphically display waveforms as well as set and control operation, all via the oscilloscope's GPIB connection. A PC needs only to point a web browser at the controller's IP address and the Java application is automatically downloaded and executed. The Java application then uses the XML-RPC protocol to control operation of the GPIB bus via the controller.



Client Software

To demonstrate and test the capabilities of the networked controller, we developed a client application. The goal of the client was to provide a familiar user interface for a GPIB device connected to our controller. In our case, this was a Tektronix TDS-210 digital oscilloscope. An application with a simple graphical user interface exposing enough functionality to perform most of the basic operations of a scope would do the trick.

There were three basic requirements of the programming language we chose to implement the client. The first was that a free and functional XML-RPC client library was available. Preferably, it would also be open source. One of the main reasons we chose XML-RPC was because of its prevalence, so this requirement didn't narrow the field much. A quick web search found free libraries for all the popular languages including, but not limited to, C/C++, Java, Microsoft .NET, Perl, Python, PHP, and TCL. The second requirement was that free and mature development tools, including a compiler/interpreter and a windowing toolkit, were available. Again, this didn't shorten the list much because it only excluded .NET from the list above. Finally, to exploit one of the primary value-added features that our device provides, it needed to be embeddable in the eZ80's HTTP server as a web browser applet. Of course, this last requirement made Java the clear choice.

In a nutshell, the client applet presents the scope screen and controls, translating user interactions into GPIB calls and screen updates. The scope's GPIB interface is like a console that accepts ASCII commands and queries. Commands and queries are sent from the client to the scope as RPCs invoking the GPIB Send method. Answers to queries are read back with RPCs invoking the GPIB Receive method. For example, to query the vertical scale (i.e. Volts/division) of channel one, the string "CH1:SCALE?" is sent with a GPIB Send RPC. To read the response to the query, a subsequent GPIB Receive is sent. The following code snippet shows these steps done in Java.

```
// Create a client object connected to our server
XmlRpcClient xmlrpc = new XmlRpcClient("http://192.168.1.51/RPC2");

// Build the gpib.Send() parameters list
Vector params = new Vector();
params.addElement(new Integer(2)); // GPIB primary address
params.addElement("CH1:SCALE?".getBytes()); // String to send (in
// binary format)
params.addElement(new Integer(1)); // Termination value

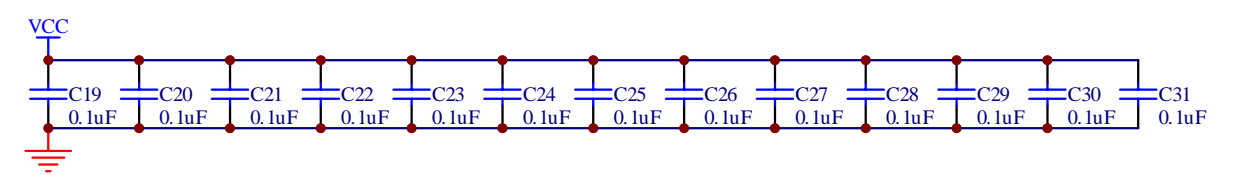
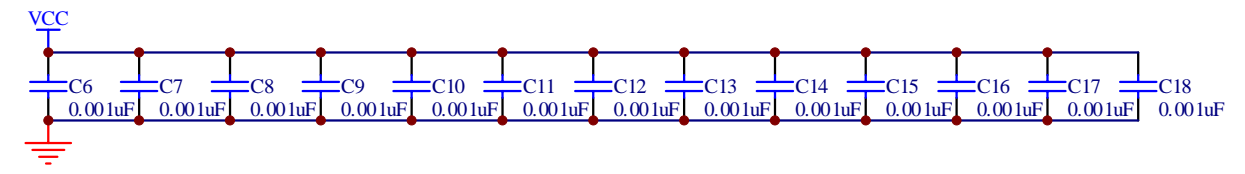
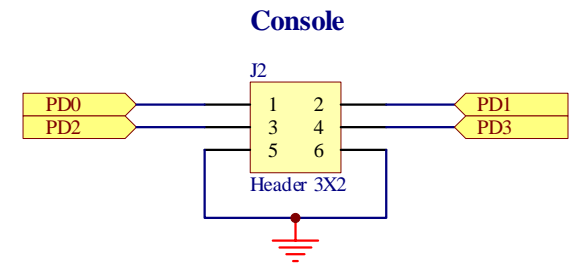
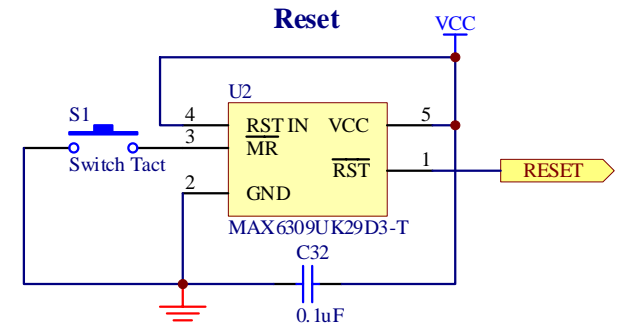
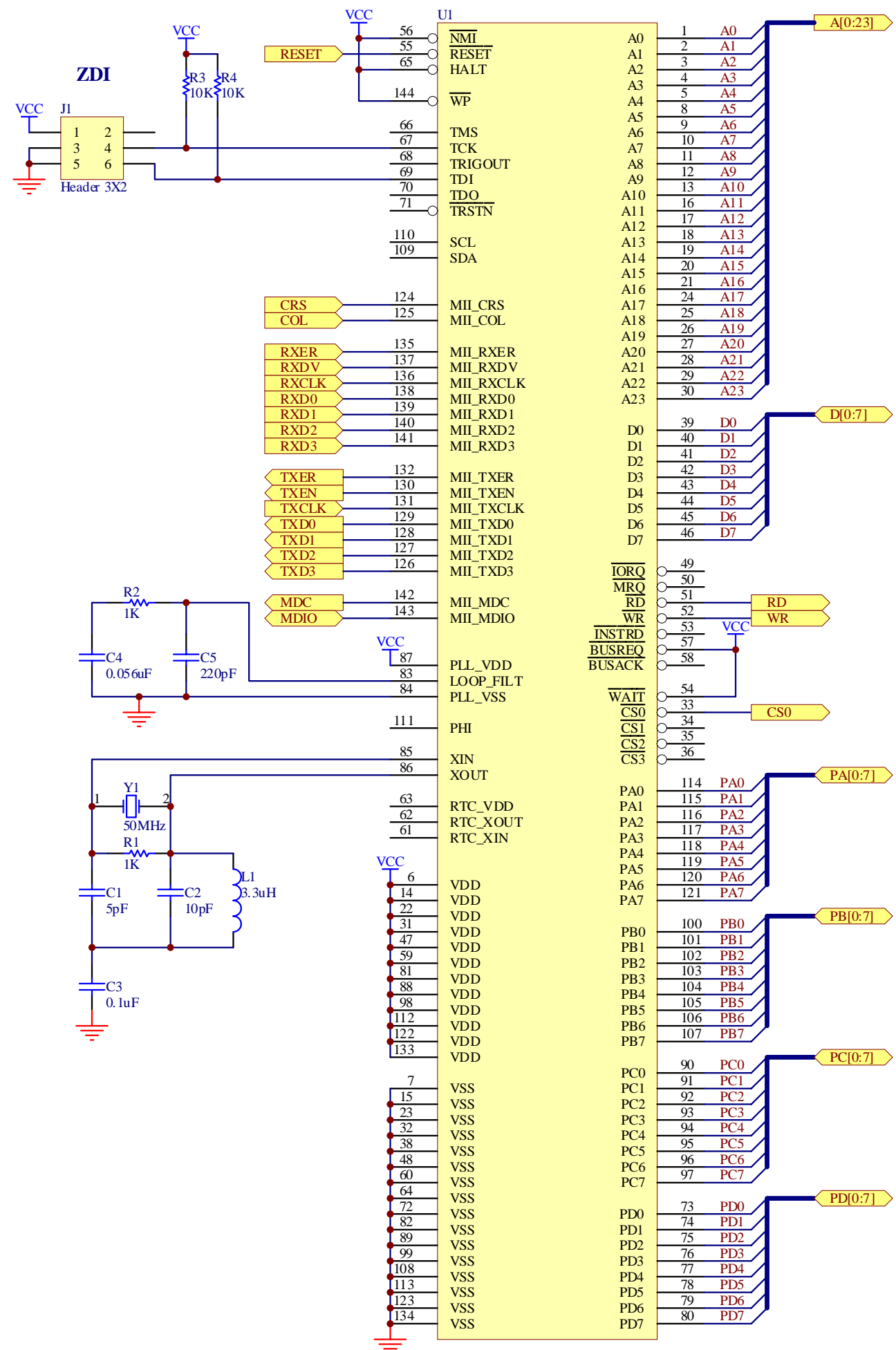
// Make the Send remote procedure call
xmlrpc.execute("gpib.Send", params);

// Build the gpib.Receive() parameters list
Vector params = new Vector();
params.addElement(new Integer(2)); // GPIB primary address
params.addElement(new Integer(500)); // Receive a maximum of
// 500 bytes
params.addElement(new Integer(1)); // Termination value

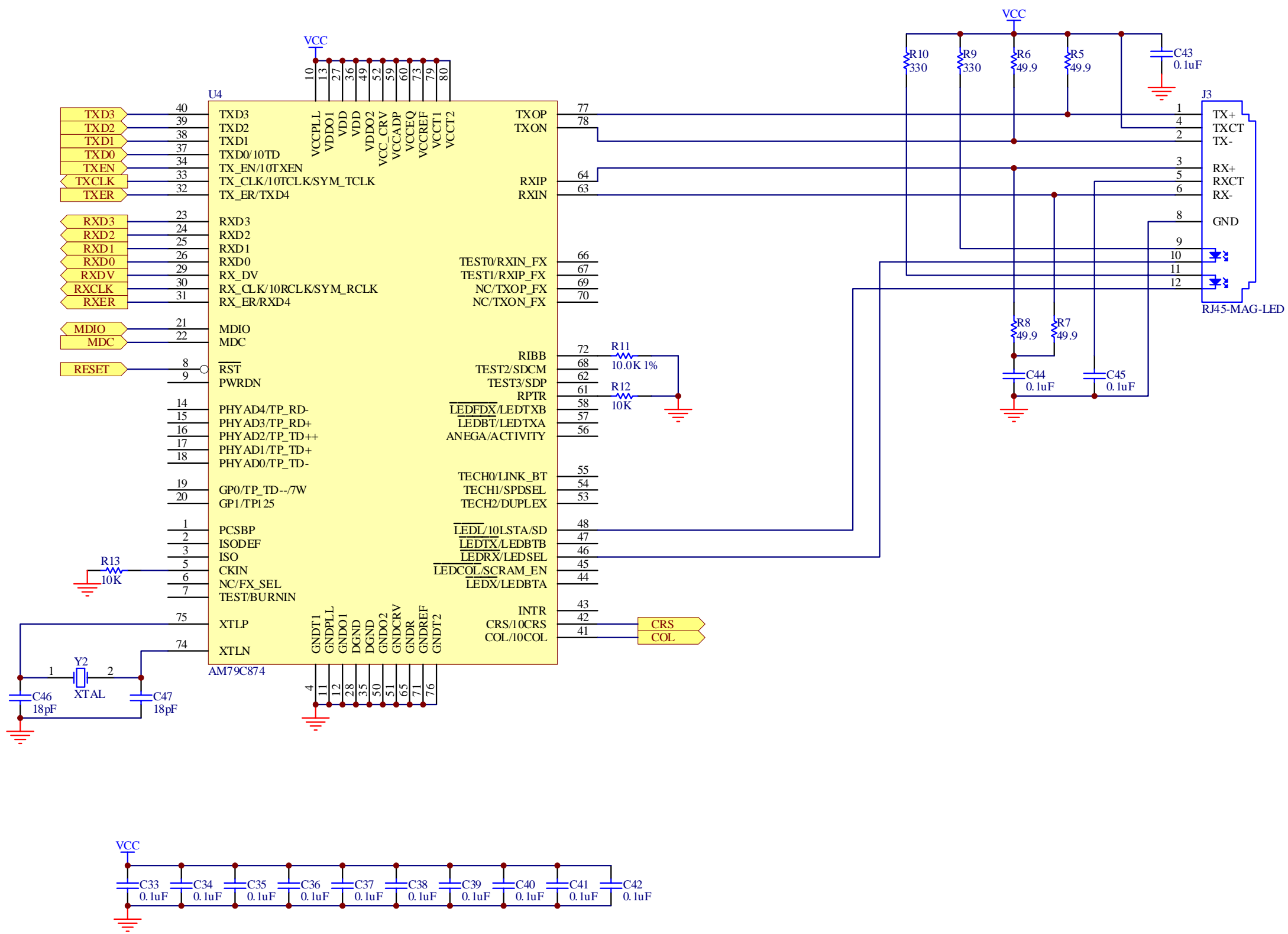
// Make the Receive remote procedure call and store the response
Hashtable response =
    (Hashtable) xmlrpc.execute("gpib.Receive", params);

// Receive returns the query response in a hash table with the key
// "buffer". Extract the binary-formatted answer and store it in a byte
// array.
byte[] buffer = (byte[]) response.get("buffer");

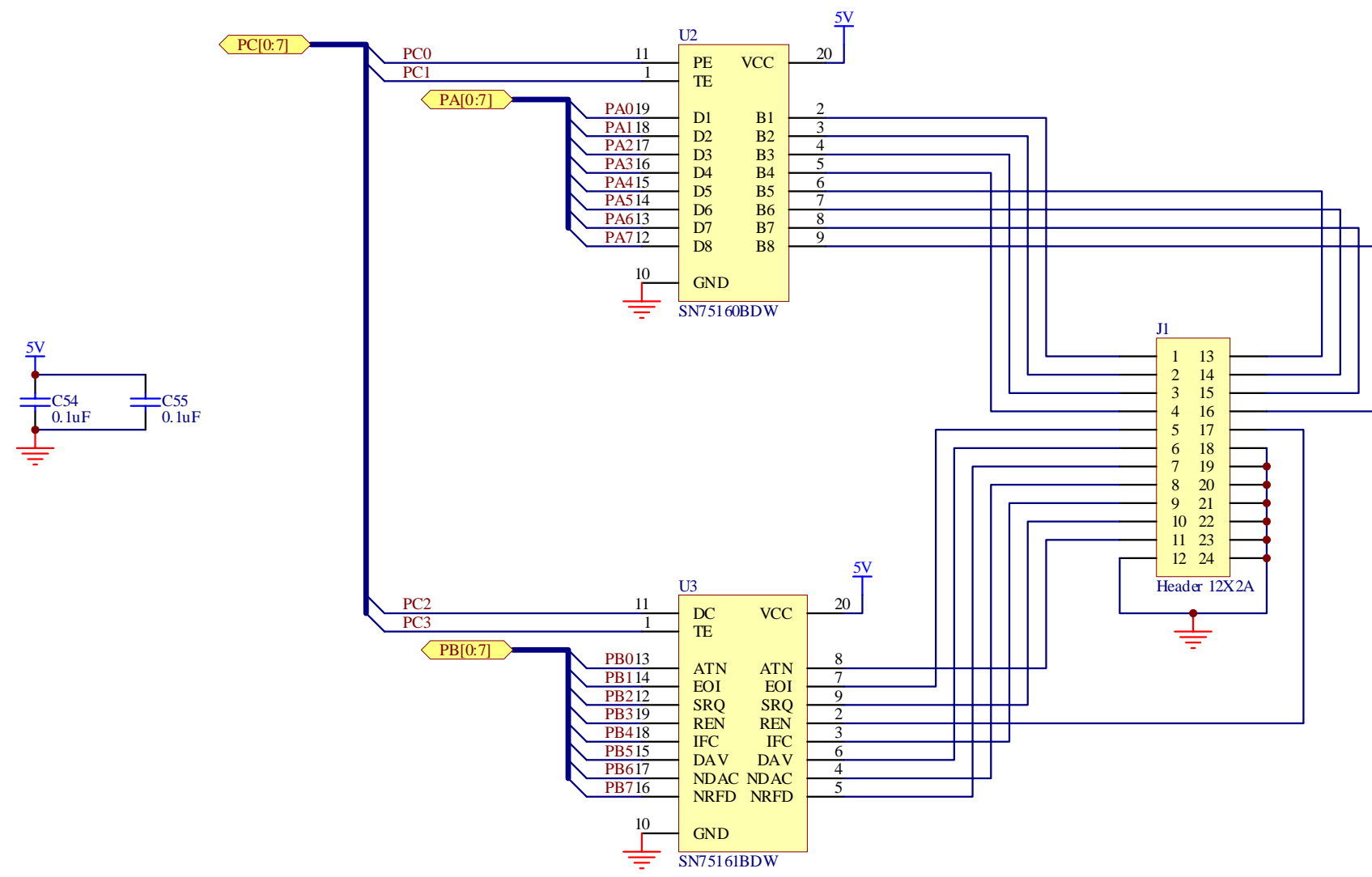
// Convert the answer back to an ASCII string. This will be the Volts/div
// setting of channel one in floating point (e.g. 100E-3 for 100 mV).
String str = new String(buffer, "US-ASCII");
```



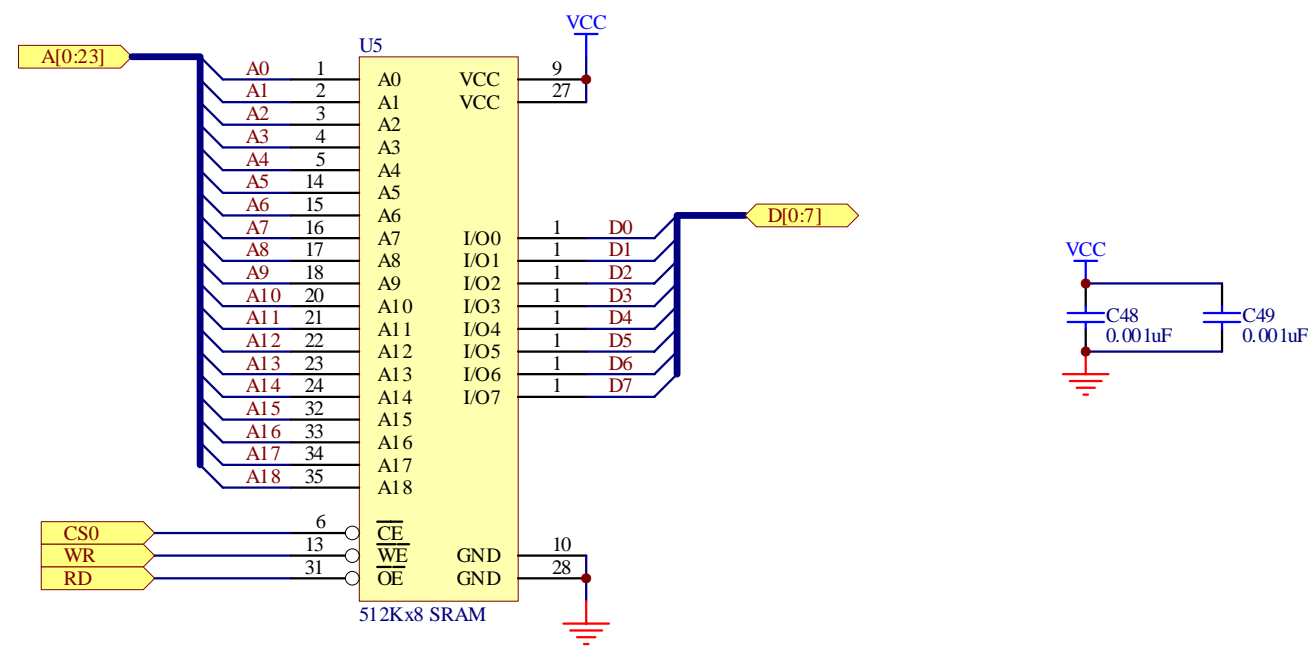
Title eZ80F91 Microcontroller			ZiLOG 2004 Flash Nets Cash Design Contest Project #: eZ2963
Size: B	Number:	Revision: 1	
Date: 9/26/2004	Time: 11:17:20 AM	Sheet 1 of 5	
File: eZ80.SchDoc			



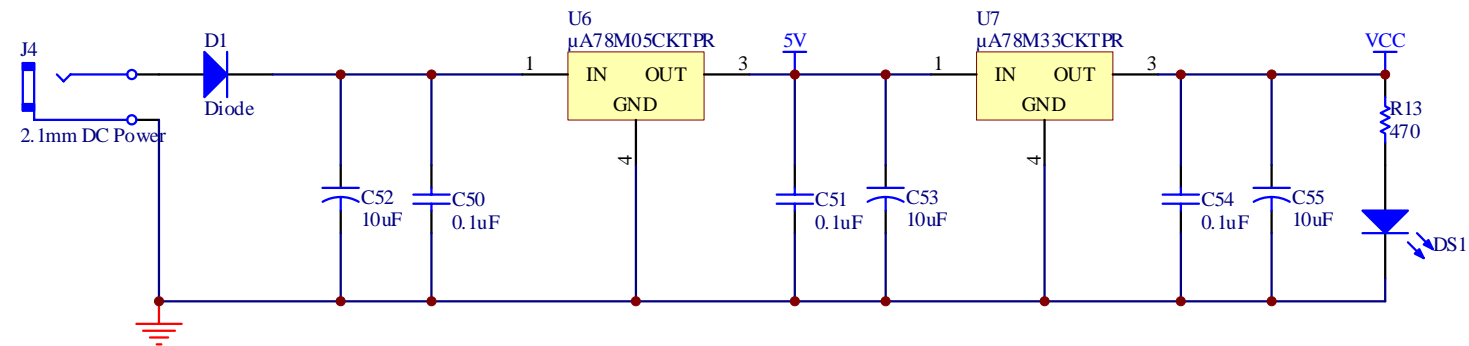
Title Ethernet PHY			ZiLOG 2004 Flash Nets Cash Design Contest Project #: eZ2963
Size: B	Number:	Revision: 1	
Date: 9/26/2004	Time: 11:17:20 AM	Sheet 2 of 5	
File: Ethernet.SCHDOC			



Title <i>GPIB Interface</i>			ZiLOG 2004 Flash Nets Cash Design Contest Project #: eZ2963
Size: B	Number:	Revision: 1	
Date: 9/26/2004	Time: 11:17:20 AM	Sheet 3 of 5	
File: GPIB.SchDoc			



Title External Memory			ZiLOG 2004 Flash Nets Cash Design Contest Project #: eZ2963
Size: B	Number:	Revision: 1	
Date: 9/26/2004	Time: 11:17:20 AM	Sheet 4 of 5	
File: Memory.SCHDOC			



Title Power Supply			ZiLOG 2004 Flash Nets Cash Design Contest Project #: eZ2963
Size: B	Number:	Revision: 1	
Date: 9/26/2004	Time: 11:17:20 AM	Sheet 5 of 5	
File: Power.SCHDOC			

